**Algorithm**

1. **Initialization:**
   * Set up the system parameters including the header files, sampling rate, baud rate, and pin configurations.
   * Initialize the communication interfaces (e.g., Serial) for monitoring and debugging purposes.
2. **Setup:**
   * Attach the servo motors to their respective pins and set their initial positions.
   * Configure the EMG sensor input pin and any auxiliary pins required for system operation.
   * Define the threshold voltage
3. **Main Loop:**
   * Continuously sample the EMG signal at the defined sampling rate.
   * Apply a band-pass Butterworth IIR digital filter to the raw EMG signal to extract relevant muscle activity within the desired frequency range.
   * Compute the EMG envelope using an envelope detection algorithm to estimate the magnitude of muscle activation.
4. **Gesture Recognition:**
   * Compare the normalized envelope value with a predefined threshold to determine muscle activation and gesture recognition.
   * Implement a hysteresis mechanism to prevent rapid toggling of the arm due to noise or minor fluctuations in muscle activity.
   * Define specific thresholds for opening and closing gestures based on individual user characteristics and preferences.
5. **Servo Control:**
   * Based on the detected gesture:
     + If the muscle activation exceeds the closing threshold:
       - Close the arm by rotating the servo motors to the predefined closed position.
     + If the muscle activation falls below the opening threshold:
       - Open the arm by rotating the servo motors to the predefined open position.
   * Implement a gesture delay to prevent rapid and unintended toggling of the claw in response to minor fluctuations in muscle activity.

Reference Github of Upsidedown Labs

**Algorithm of EMG Band Pass Filter**

Algorithm for the Band-Pass Butterworth IIR digital filter:

1. **Initialization**:
   * Initialize the state variables **z1** and **z2** for each filter section to zero.
2. **Filtering Process**:
   * For each input sample:
     + For each filter section:
       - Calculate the intermediate variable **x** using the difference equation of a second-order IIR filter.
       - Update the output using the calculated **x** value and the previous state variables.
       - Update the state variables **z1** and **z2** for the next iteration.
3. **Output**:
   * Return the filtered output.

Here's a breakdown of the steps within the filtering process:

* For each filter section:
  1. Calculate the intermediate variable **x** using the difference equation:

x = input - a1 \* z1 - a2 \* z2

where **input** is the current input sample, **z1** and **z2** are the previous state variables, and **a1** and **a2** are the filter coefficients.

* 1. Update the output using the calculated **x** value and the previous state variables:

output = b0 \* x + b1 \* z1 + b2 \* z2

where **b0**, **b1**, and **b2** are the filter coefficients for the output.

* 1. Update the state variables **z1** and **z2** for the next iteration:

z2 = z1 z1 = x

Repeating these steps for each input sample, obtained the filtered output of the Band-Pass Butterworth IIR digital filter.

**General Difference Equation**

The code implements a Band-Pass Butterworth IIR digital filter using second-order sections (biquads). Each biquad represents a second-order IIR filter section. Let's break down the mathematical expression for each biquad.

The general difference equation for a second-order IIR filter is:

*y*[*n*]=*b*0​*x*[*n*]+*b*1​*x*[*n*−1]+*b*2​*x*[*n*−2]−*a*1​*y*[*n*−1]−*a*2​*y*[*n*−2]

Where:

* *y*[*n*] is the output at time *n*
* *x*[*n*] is the input at time *n*
* *x*[*n*−1] and *x*[*n*−2] are the previous input samples
* *y*[*n*−1] and *y*[*n*−2] are the previous output samples
* *b*0​,*b*1​,*b*2​ are the feedforward (numerator) coefficients
* *a*1​,*a*2​ are the feedback (denominator) coefficients

Let's express each biquad in the code as difference equations:

**First Biquad:**

z1\_1[n] = x[n] - 0.05159732 \* z1\_1[n-1] - 0.36347401 \* z2\_1[n-1] z2\_1[n] = z1\_1[n-1] y\_1[n] = 0.01856301 \* z1\_1[n] + 0.03712602 \* z2\_1[n] + 0.01856301 \* z2\_1[n-1]

**Second Biquad:**

z1\_2[n] = y\_1[n] - (-0.53945795 \* z1\_2[n-1] - 0.39764934 \* z2\_2[n-1]) z2\_2[n] = z1\_2[n-1] y\_2[n] = 1.00000000 \* z1\_2[n] - 2.00000000 \* z1\_2[n-1] + 1.00000000 \* z2\_2[n-1]

**Third Biquad:**

z1\_3[n] = y\_2[n] - (0.47319594 \* z1\_3[n-1] - 0.70744137 \* z2\_3[n-1]) z2\_3[n] = z1\_3[n-1] y\_3[n] = 1.00000000 \* z1\_3[n] + 2.00000000 \* z1\_3[n-1] + 1.00000000 \* z2\_3[n-1]

**Fourth Biquad:**

z1\_4[n] = y\_3[n] - (-1.00211112 \* z1\_4[n-1] - 0.74520226 \* z2\_4[n-1]) z2\_4[n] = z1\_4[n-1] y\_4[n] = 1.00000000 \* z1\_4[n] - 2.00000000 \* z1\_4[n-1] + 1.00000000 \* z2\_4[n-1]

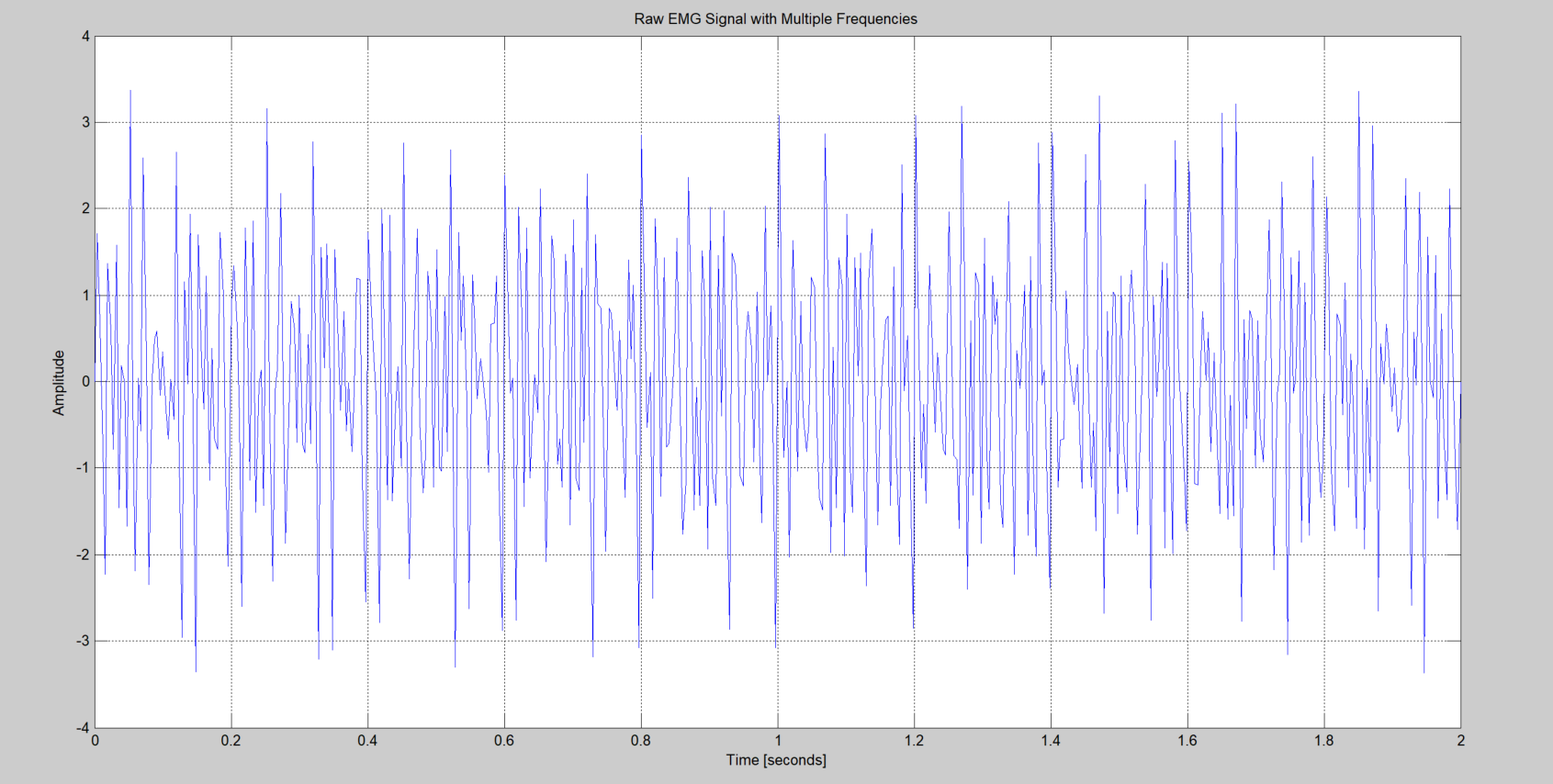
Where:

* *z*1*i*​[*n*] and *z*2*i*​[*n*] are the state variables for the *i*-th biquad at time *n*
* *x*[*n*] is the input at time *n*
* *yi*​[*n*] is the output of the *i*-th biquad at time *n*

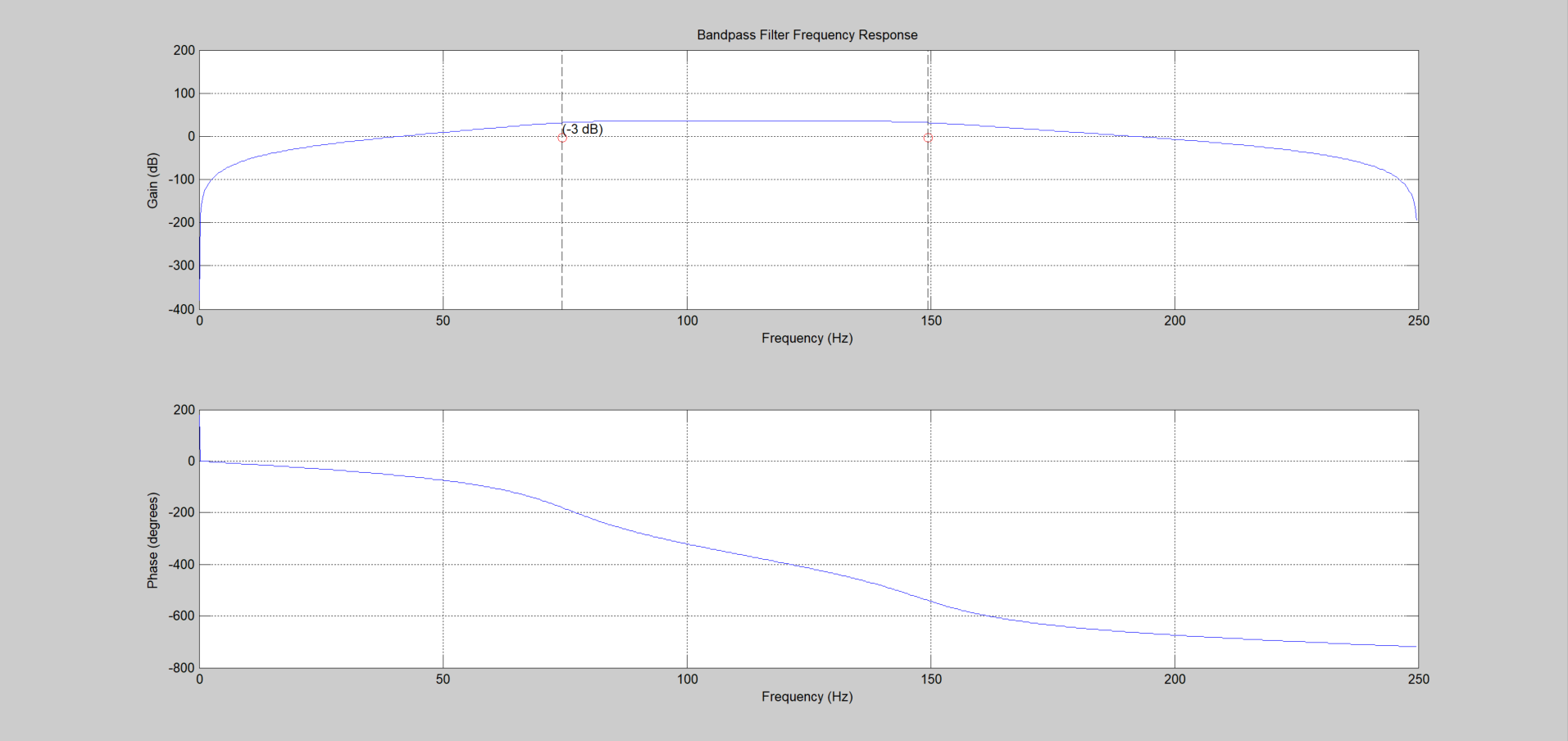
This set of equations describes the mathematical expression for the given Band-Pass Butterworth IIR digital filter implemented in the provided code. Each biquad contributes to the overall filter response, and the output of one biquad serves as the input to the next biquad in the chain.

**Simulation of BPF in MATLAB**

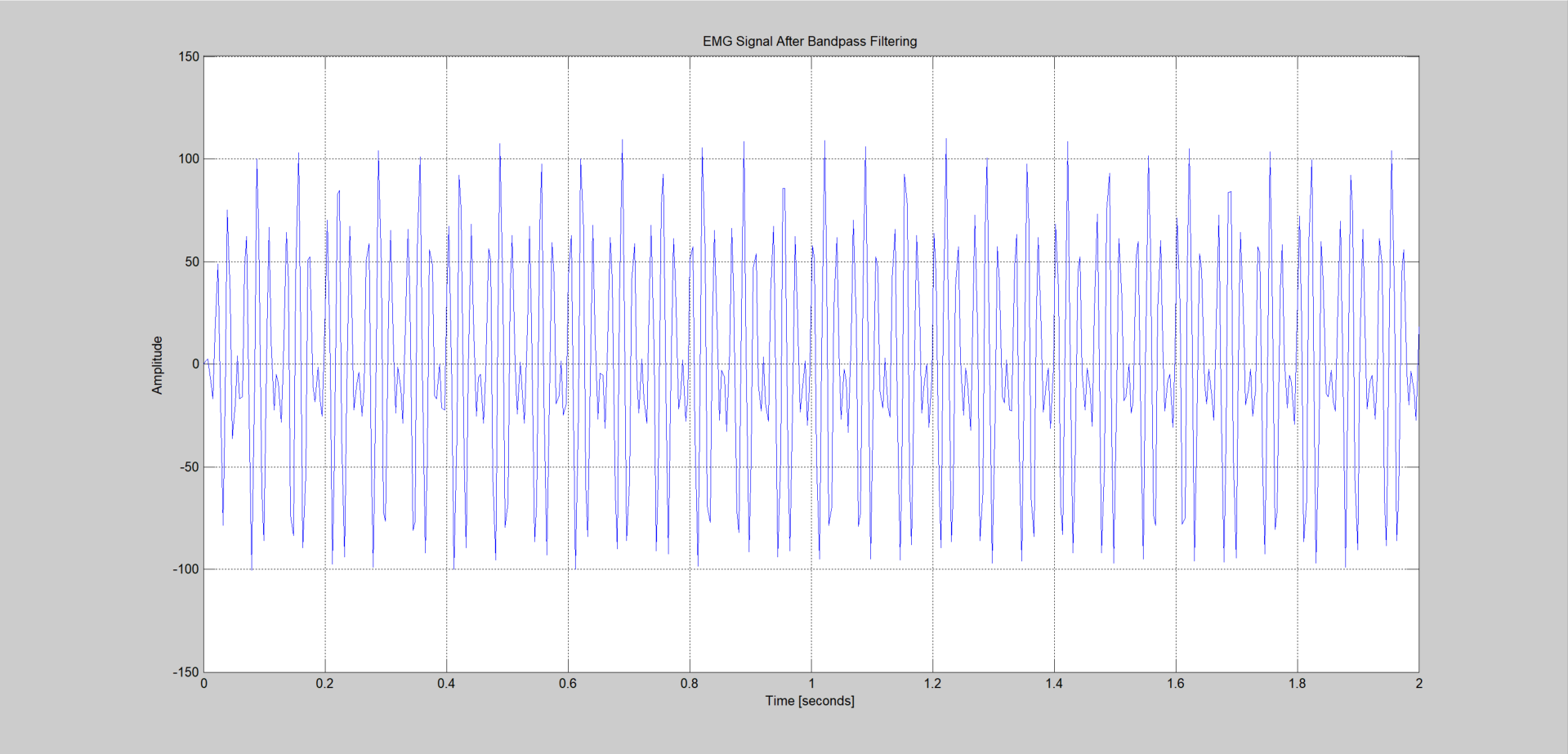
Raw EMG signal with multiple frequencies of 45Hz, 60Hz, 90Hz, 100Hz and 160Hz



Frequency response of the filter with -3db point



Filtered EMG Signal after Band Pass Filter consisting of frequencies 90Hz and 100Hz



**Reference:**

<https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.butter.html>

<https://courses.ideate.cmu.edu/16-223/f2020/Arduino/FilterDemos/filter_gen.py>

**Algorithm for Envelope Detection-**

1. Subtract the previous EMG signal value from the sum.
2. Add the absolute value of the current EMG signal to the sum.
3. Store the absolute value of the current EMG signal in the circular buffer at the current index.
4. Update the data index to point to the next position in the circular buffer, wrapping around to the beginning if necessary.
5. Compute the average of the EMG signal values in the circular buffer by dividing the sum by the buffer size.
6. Multiply the average by 2 to scale the envelope signal.
7. Return the computed envelope signal.

**EMG Signal After Filter With The Detected Envelope**

